Flight Software Test Initiatives at NASA/GSFC

Maureen Bartholomew
NASA/GSFC
code 582

Agenda

- Summarize current test innovations
- Highlight top concerns for flight software testing
- Suggest future test initiatives

Current Test Innovations at GSFC

- Management of FSW tables
- Use of Autocode generation tool for flight code of attitude control subsystem algorithms.
- Automated tools for plotting attitude & orbit subsystem data

Current Innovative Test Initiatives - Management of FSW Tables

- Use of databases for flight software tables
  - Define and retrieve information reliably by all
  - Link flight software elements together
  - Link flight software variables with others such as subsystem GSE and ground systems
  - Under CM - each variable assigned to a single person/owner
  - automated population of flight code from the database (eliminates potential translation/cut & paste problems)
- Definition of flight software tables on the ground system.
  - All tables defined using same definition as commands and telemetry
  - Table editing and reloading as part of STOL test procedures
    - Dump FSW table
    - Make table changes mnemonically via the ground system
    - Reload new table
    - Display of labeled formatted FSW tables through ground system page displays (Automatic page display generator also available).

Current Innovative Test Initiatives - Management of FSW Tables

- Examples of database applications
  - Packet Routing Tables
    - All packet routing requirements stored in database
    - Database can generate all header, and table initialization files
  - 1773 Bus schedule table generator
    - Generates schedule based on timing requirements and list of constraints
  - Telemetry & Statistic Monitoring Tables
    - TSM requirements are stored in database
    - Uses ground system database to resolve mnemonic telemetry locations
    - Database software can generate FSW TSM tables
  - Relative Time Sequences
    - RTS information is stored in database
    - Database software can generate FSW RTS loads
    - Linked to Telemetry & Statistic Monitors
  - Telemetry Output and Data Storage filter tables
    - Telemetry sample and filtering rates stored in database
    - Database software can generate all FSW filter tables
  - ACS Database Tool
    - Each flight parameter assigned to a single individuals for flight values (password protected)
    - Generates ACS Flight software tables
    - Generates scripts for hybrid dynamic simulator (HDS) initialization
    - Generates scripts for Hifidelity simulations

Current Innovative Test Initiatives - Autocode

- Autocode generation from ACS analysis tools.
  - Traditional method:
    - ACS Analysts at GSFC would develop Hi Fi simulations in Fortran and would interface with the FSW team through a handwritten algorithm document.
    - The algorithms would be translated into flight software.
  - Innovation
    - The MAP controls engineers are using ISI's MATRIXx for their controls analysis. Provides:
      - Graphical analysis environment
      - Autonomous code generation facility (AutoCode™)
    - The 'C' code generated from AutoCode is being used successfully in the FSW load
  - Result
    - no translation required. Reduces potential errors. Analyst-tested flight code!
    - forced hi-fidelity simulation to have a common design with FSW (better unit test data)
    - Greatly improves turn-around time related to algorithm changes during development (creates FSW test efficiencies).

Current Innovative Test Initiatives - Automated Plotting

- Automated Plotting Tools
  - Traditional method
    - A flight software tester would develop a test and execute it on the hardware.
    - An attempt would be made to duplicate the initial conditions for the hi-fidelity (HIFI) comparison run.
    - The output from both would be merged and plotted by the FSW test engineer in his/her favorite plotting package (each person using his/her favorite units and scaling).
    - Innovative method
    - Develop tools to automatically scan FSW test data for critical data (e.g. mode transitions, command quaternions)
    - Critical data used to create script file for HIFI run which matches FSW run
    - Defines predefined comparison plots between FSW, HDS and Hi-fidelity sim
  - Result
    - Easily make comparison plots between FSW, HDS and Hi-fidelity simulation
    - Standardized plots - Reduces plotting and test review time!

Current Innovative Test Initiatives
General

- Used personnel with various expertise to create a well-rounded test team
  - space controllers
  - flight software maintenance personnel
  - attitude control analysts
  - flight software testers
- Had a Flight Software Manager responsible for ALL flight software.

Top Concerns for Flight Software Testing

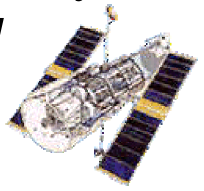
- Ensuring test coverage
  - Requirements
  - Operations/Systems Scenarios
  - Contingency/Failure Scenarios
- Defining/Ensuring proper test configuration at all times
  - have proper GSE
  - definition of test setup
  - Known FSW versions, FSW database values, simulator values, ground system database

New FSW Test Initiatives

- Continue to use current innovations described
- Develop tools to standardize requirements definition
  - include "use" cases
  - include fault cases

Goddard Space Flight Center

HST Payload FSW

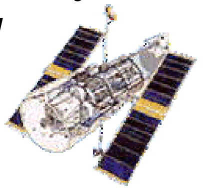


HST Payload Flight Software Testing

R. Koehler

Computer Sciences Corporation

HST Payload FSW



Existing Test Processes

- Existing processes differ for two software teams:
  - NSSC-I
    - overall payload commanding and monitoring
  - Science Instrument
    - one or more systems (80386) embedded within each science instrument
- Both teams drive tests from a copy of the same ground system used to control the telescope
  - has scripting ability
  - provides familiarity when called to the control center

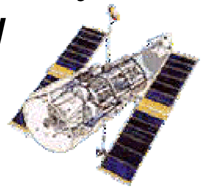
NSSC-I Test Processes

- Formal unit tests use NSSC-I software simulator
- System testing uses NSSC-I GEM
- Acceptance test dry run uses NSSC-I GEM
- Formal acceptance test uses NSSC-I GEM, includes system and regression testing
- Delivered FSW tested for installation and interfaces by operations support groups using NSSC-I GEM or NSSC-I flight spare with scripts developed by those groups and/or STScI

Goddard Space Flight Center

SI Test Processes

HST Payload FSW

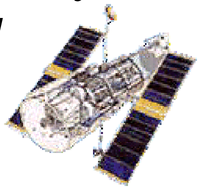


- Occasional algorithm test on any platform
- Unit test uses Kontron in circuit emulator in software bench
- Integration test uses 80386 in software bench, FSW loaded to volatile RAM
- System and regression test uses 80386 in software bench, FSW loaded to EEPROM, scripts developed by STScI
- Additional testing performed by operations support groups, developed by them or STScI
- Post installation testing by STScI

Goddard Space Flight Center

Top Technical Problem Areas

HST Payload FSW

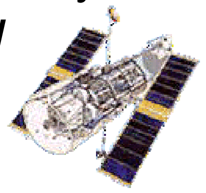


- Lack of automation, tools
- Documentation requirements have not always been reviewed against audience needs
- Incomplete/inaccurate simulations
  - NSSC-I software simulator has limited modeling of timing
  - NICMOS software bench cannot ingest test images
  - STIS software bench does not simulate power changes in response to some error conditions
  - NSSC-I test interfaces to partial simulations
  - MASIS design assumes one data base

Goddard Space Flight Center

Top Technical Problem Areas, cont.

HST Payload FSW

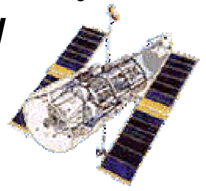


- Test suites are not modular and re-usable
  - Test systems contain different implementations of similar functions
    - SIMREQ vs. TAC
    - /G command vs. /R commands

Goddard Space Flight Center

Tools and Automation

HST Payload FSW

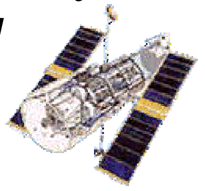


- No tools to determine test paths
- Some existing tools were poorly developed
  - several thousand lines of uncommented, undocumented Fortran IV
  - simulation software which cannot be rebuilt from delivered source
- NSSC-I unit tests drivers must be manually created
- NICMOS unit tests require manual creation of test driving code which must embed any desired test science images
- No automatic interpretation of test results
  - generic report for each data type, tester manually scans to check relevant data

Goddard Space Flight Center

Tools and Automation, cont.

HST Payload FSW



- Manual process verifies SI difference loads map to code changes
- Manual process verifies SI ground vs. spacecraft s/w image differences
- Test components located on a variety of systems, requiring manual steps to copy data to systems where existing tools are located
- Test instructions not centrally located
- Command Loader meets operational needs, awkward and little support for our use

HST Payload FSW

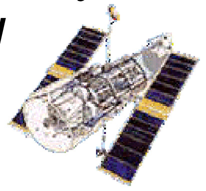
Futures

- VAP analysis system is being developed to automate verification of test results
  - developed as a set of Java applications so it can be run where the data resides
- WWW based centralization of documentation and test instructions (http://hstplsrv.gsfc.nasa.gov)
- VAP - CCS integration tests were developed to be modular and reusable
  - reusing during port of NSSC-I tests from PRS to CCS
    - ACS and ASCS-NCS NSSC-I AP acceptance tests
    - NSSC-I GEM and related hardware checkout

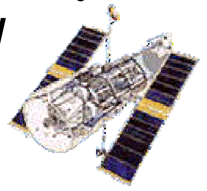
Goddard Space Flight Center

Futures, cont.

HST Payload FSW



- Documentation has been and should continue to be streamlined to meet audience needs
- New tools are professionally developed
- New tool requirements developed with the NSSC-I and SI teams



Acronyms

ACS	Advanced Camera for Surveys
ASCS-NCS	Aft Shroud Cooling System - NICMOS Cooling System
AP	Applications Processor
CCS	Control Center System
EEPROM	Electrically erasable programmable read-only memory
GEM	Ground Equivalent Model
MASIS	Monitor and Science Instrument Simulator
NICMOS	Near-Infrared Camera and Multi-Object Spectrometer
NSSC-I	NASA Standard Spacecraft Computer Model-I
PORTS	Payload Operations Real-time System
PRS	PORTS Refurbishment System
STIS	Space Telescope Imaging Spectrograph
VAP	Verification and Acceptance Program

HST Flight Software SSM486 Test Effort

- Current Status
  - Build and System Level Testing Completed On-Schedule
    - Confidence Tests/Servicing Mission Preps Underway
  - Transitioning to Maintenance Phase of Software Life-Cycle
    - Mature Development Test Tools to be used during Maintenance

- Vision for Future
  - To Provide the same High Standard of Flight Software Testing as Established during Build and System Testing

Development Environment

- **Development Team**
  - Co-located in the Nations Bank Building, with Test and Tool teams.
- **Development**
  - PC-based
  - Shared network drive contains development team files
    - drive backed up daily
- **Design**
  - *System Architect* - a COTS Computer Aided Software Engineering (CASE) Tool.
- **Configuration Management**
  - *PVCS* - a COTS Version Management System for the PC.
- **Unit Testing**
  - SIM486 / FVS486 / FSS486 - In-house developed Simulation tools for testing SSM software.
- **Integration Testing**
  - NBB Lab (first floor)
    - HST486 Brassboard
    - HST486 Test Set - PC-based
    - Microtek Intel 80486 Emulator - PC-based
    - CCS Ground System - SG/PC-based
    - PCS Simulator - DEC Alpha-based (PC-based in future)

Process Overview

- Build Development, During the Build, Each Developer:
  - Performs a **Peer Review** of the Detailed Design for the Unit. The Reviews Include Detailed Pseudo Design Language (PDL), Supporting Structure Charts, C-Language Header Files and Requirements Checklists. Invite Appropriate Build Test Team Member(s) to the Peer Review.
  - Performs a **Code Review** for the Unit. The Reviews Include the Updated Peer Review Material Plus Code Listings. Again, Build Test Team Member(s) Attend the Code Reviews.
  - Performs **Unit Testing** of the Unit. A Final Unit Test Report Is Delivered to the Flight Software Lead.
- Integration Testing
  - Perform a Composite Build of All Units.

    Test the Software As a Whole on the SSM486 Hardware

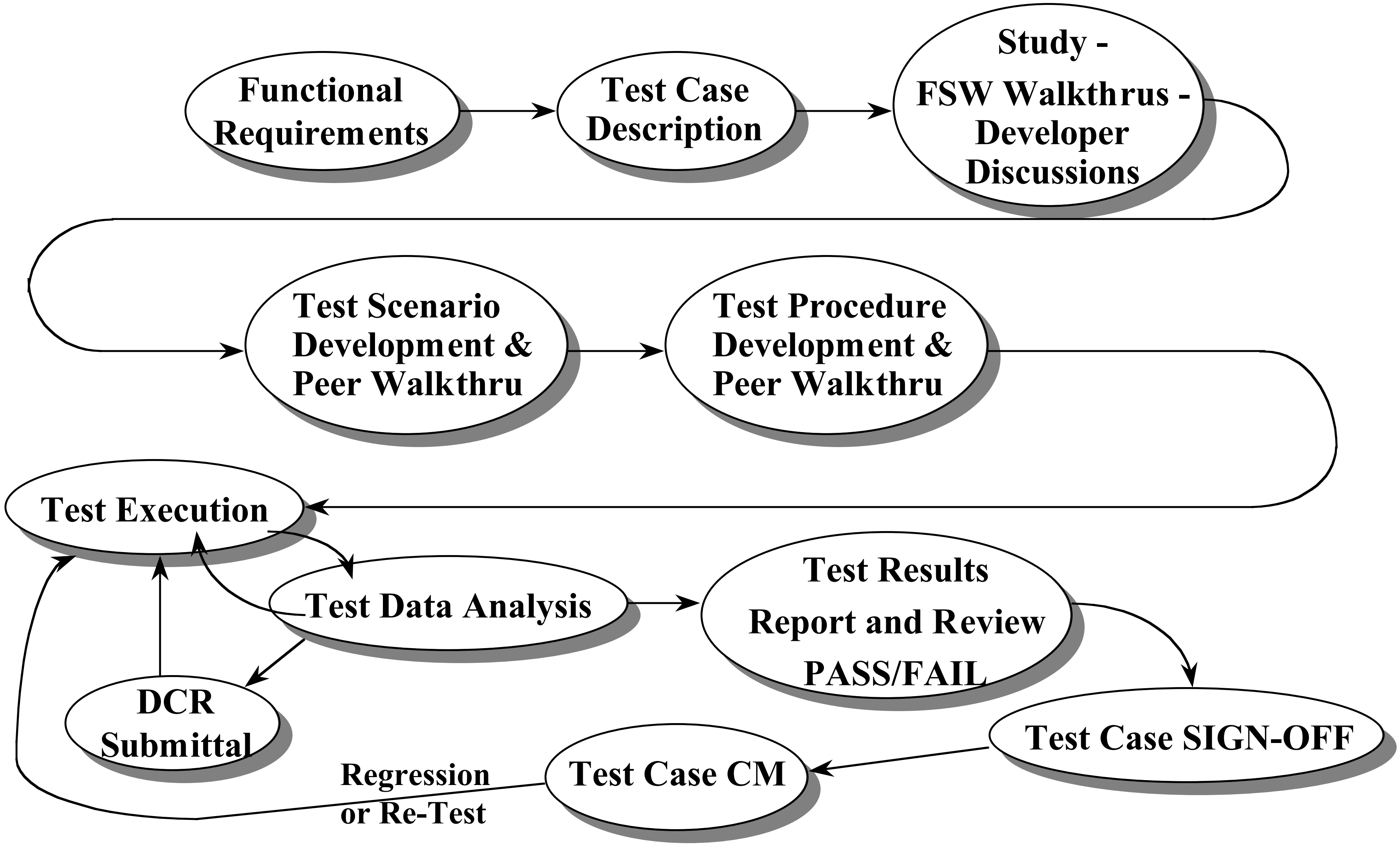
Process Overview

- Build Testing, During the Build Test Cycle Each Tester:
  - **Reviews** the Functional **Requirements** to Be Verified
  - Performs **Scenario Walkthrough** of Specific Test, Showing Requirements to Be Verified and Layout of Test,

    Analogous to Design Peer Review
  - Conducts Test **Procedure Walkthrough**,

    Showing Details of Test and Dry Run Results
  - Performs **Formal** Build and Regression Test **Execution**
  - Provides Analysis and Documents Results in **Test Report**
  - **Archives Test Results** for Future Regression Testing
  - Updates Requirements Traceability (Using RTM)
- System Testing
  - Process Same as Build Testing, Focus is on:
    - Operation Scenarios Using CCS and Simulators
    - End-to-End System Testing

Build/System Test Process


Functional Requirements
Test Case Description
Study - FSW Walkthrus - Developer Discussions
Test Scenario Development & Peer Walkthru
Test Procedure Development & Peer Walkthru
Test Execution
Test Data Analysis
Test Results Report and Review PASS/FAIL
DCR Submittal
Test Case SIGN-OFF
Test Case CM
Regression or Re-Test

Problems/Issues That Affect Cost/Schedule

- Requirements Definition
  - Late, Vague, Untestable Requirements
    - Addressed these by delaying development of some features until later Builds. This caused schedule problems, but no technical problems.

- Simulation and/or Lab Development Parallel with Test Effort
  - Test Platforms need to be mature by start of software testing
    - Test Tool development anticipated FSW Build features, "just-in-time" maturity
    - Testers ended up testing the Test Tools also, lots of Test tool bugs found.

- These problems largely mitigated for SSM486 by co-location of the complete FSW team allowing excellent communications to solve problems quickly.

Reducing SW I&T Costs

J. Troeltzsch

10/4/99

Decisions During Design Have Large Impact on I&T

- Problem: Requirements are levied without understanding the true cost associated with I&T and ground system support.
- Example: A single added observing mode on HST or SIRTF created the need for >600 of hours of additional engineering labor, associated cost > $75,000.
- Proposed Solution:
  - Full life cycle cost estimation and accounting as part of software requirements review.
  - Systems engineering should scope requirements to match budget.

Don’t Build It If You Won’t Use It

- Problem: Large portions of flight software are never used on orbit. Most of these fall into two categories: “neat ideas” and contingency operations. Most unused code equates to wasted money.
- Examples: Approximately 50% of GHRS software was never used on orbit. <1% of the errors defined and tested in the NICMOS code have actually occurred during ground test or on orbit.
- Proposed Solution:
  - Severely restrict operational modes to support the majority of the users.
  - Establish a probability of occurrence for errors and focus resources on most likely errors. Lower probability errors should have minimal, simple code associated with them.
  - Systems engineering should strive to reduce complexity.

Automated Design Gains Can Reduce I&T Costs

- Problem: Large amount of the life-cycle cost is in the preliminary and detailed design. This effort often overruns the budget and schedule. This reduces the time to code and integrate and increases I&T cost.
- Example: Virtually every new space based real time embedded system is late and over budget.
- Proposed Solution:
  - Use automated software CAD system to design, document, and simulate the embedded system.
  - Use same tool to generate the code. SIRTF used ObjectGeode with good results, but did not finish simulation or code generation phase. Hard to get today's designers to buy into automated tools.
  - Simulation forces developers to get working interfaces.
  - Code reviews look at code AND simulation results.
  - Changes/fixes during I&T get made in design not directly in the code.

Automated Testing Is Happening Today

- Problem: Full path testing is absolutely necessary but very expensive and very boring.
- Example: SIRTF instrument code unit testing took 6 people 6 months. Writing test plans is tedious, busy work.
- Proposed Solution:
  - Fully automate the unit test phase. SIRTF is using DejaGnu to generate and run automated unit tests.
  - The tool reads the code to determine the needed tests and generates script code to run the test.
  - It is flexible and portable (independent of host and target).
  - Evolution of this approach should save lots of money and produce a more reliable product.

Custom Interfaces Drive Up I&T Costs

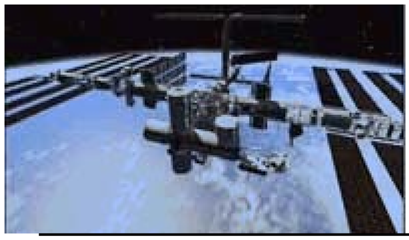
- Problem: Target processors often require custom interfaces.
- Example: HST and SIRTF processors require special hardware and software to download and run software tests.
- Proposed Solution:
  - Force target processors to have standard network interface. SIRTF used a commercial power PC with a serial and Ethernet interface for development and test prior to porting to flight card.
  - The flight card should have standard network capability built in.
  - The real-time operating system should have built in drivers for the processor interface

Put Basic Tasks Into Firmware

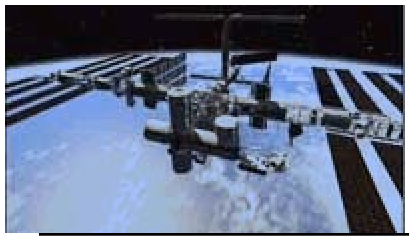
- Problem: We write, test, and maintain lots of code to perform basic functions, like run detectors and handle I/O. This provides unneeded flexibility which increases cost.
- Example: NICMOS detector control is fully programmable but no one would be crazy enough to change it on-orbit.
- Proposed Solution: Put repetitive interface control into field programmable gate arrays.
  - FPGAs are fast and reliable, reduce flight processor loading.
  - FPGA design must be done correctly and very thoroughly tested.
  - Once complete they cost nothing to maintain on orbit.
  - Locking capabilities into firmware removes temptation to add new capabilities.
  - Need to get FPGA design under software engineering control.

Flight/Ground Database Integration Needs Automation

- Problem: Command, telemetry, and error definitions are separately defined in flight and ground software. Integration often uncovers major problems. Takes twice as long to build new system since both flight and ground files must be built and checked.
- Example: SIRTf software team did not integrate actual telemetry and error handler interfaces until well after unit test. Major problems found in flight and ground designs.
- Proposed Solution:
  - Use the exact same source file for ground and flight database information. Should include command, telemetry, error, and patchable constant/symbol definition.
  - Automate generation and distribution of the files.

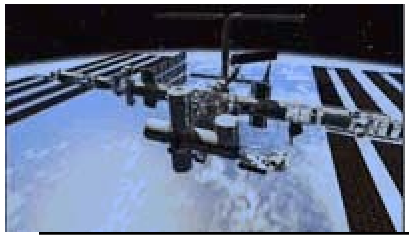


K.C. Broyles/OD
Space Station Program Office - Avionics
Stage Verification



Agenda

- Top Problems which dramatically affect cost and schedule
- Necessities to accelerate testing process compared with current methods/systems
- Vision for next generation processes/tools for flight software testing
  - "Broyles Vision"

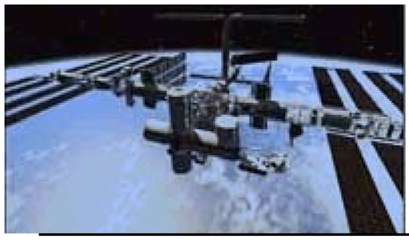


Top Problems

- Requirements
  - Accurate
  - Timely
  - ICD's
- Test Facilities and Flight Avionics lacking adequate ground support capabilities
  - Repeatable/scriptable system level facilities

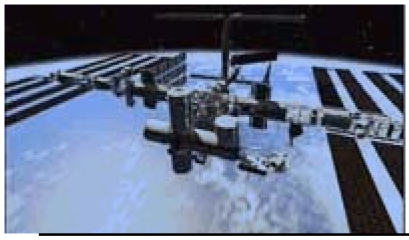
    i.e. Flight Units capable of check point

    Verifying S/W objectives is schedule constrained (gas filling a volume)



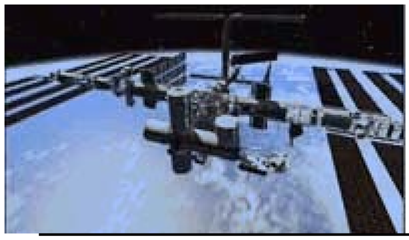
Testing Process Acceleration

- Well defined process up front
  - Entire Program ‘buy in’ on process
- Functional Requirements and Qualification requirements defined and baseline together
  - Section 3’s & Section 4’s developed together
    - exposes costs up front
  - System Spec - Segment Spec - PID's - SRS’s
- Integrated Program wide accessible system
  - Functional flow down from System Spec-SRS’s



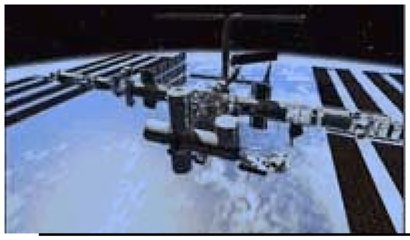
Next generation processes/tools ‘Broyles Vision’

- Flight Software testing is based on requirements
  - Regardless of how automated the test system, test objectives are derived from test requirements having stated success criteria
  - Automated test system is not total answer
    - Engineering analysis required to determined a balanced set of test objectives (cost - technical - schedule)
  - Implementation separated from requirements
    - Functional requirements traceability ends at SRS’s



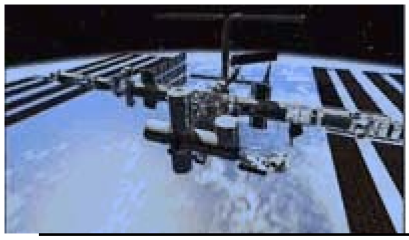
Driving Principals for Requirement generation and maintenance

- Integration is key element in today’s large distributed systems
  - Accurate/Timely communication is driving goal
  - Requirements is means for communication
- Accountability delegated downward
  - Appropriate level having responsibility of requirements
- Change Processing overhead does not influence decisions on requirements
  - i.e. cost of processing changes negligible
- Documents become a snapshot of requirements at a point in time



Next Generation Program System
'Broyles Vision'

- System - Segment - PID's - SRS's developed and maintained under Program wide system
  - Functional decomposition and traceability maintained within system
    - COTS systems exists for traceability tool (RTM)
    - System must be carried to current technologies to reach Program wide acceptance and control
      - » i.e 'Active Desktop'
  - All requirements maintained with keys
    - Version controls, Changes Pending, History rational, etc..
      - » Current "books" loose rational history



Next Generation Program System 'Broyles Vision'

- Each requirement "shall" has associated properties for development and control
  - Properties include:
    - Requirement Owner Rep., Budget authority Rep., Safety Rep., Verification Rep., CM Rep., etc...
    - Upon 'sign-off' of all Reps' in req. trace - requirement is baseline
  - Requirements 'roll' as determined by functional flow down
    - ***No longer are requirements talked in terms of 'book revisions' but rather slices in time***
- Authority is delegated down to 'owner's
- System consistently evolves to a point that requirements accurately define as built
  - Result is accurate - testable requirements

Flight Software Development and Test Methodology at JPL

Flight Software Test Workshop
Johns Hopkins University Applied Physics Laboratory
September 21-22, 1999

Daniel Eldred

Jet Propulsion Laboratory

Pasadena, CA

Outline

- Current flight software development (Deep Space One)
  - Description, background
  - Development and testing cycle
  - In-flight experience
  - Lessons learned and problem areas
- Future flight software development (Mission Data System)
  - Source code instrumentation
  - Test management systems
  - Problem reporting and tracking
  - Test case generators
  - Source code analyzers
  - Test result analyzers
- Summary and Conclusions

Deep Space One (DS1) Mission Summary

- First Launch of NASA's New Millennium Program
- Cheapest JPL deep space mission to date ($152M)
  - New experience with faster/cheaper missions for JPL
- Ambitious objectives
  - 12 new technologies (13 planned)
    - Ion propulsion
    - Autonomous on-board navigation
  - 1 asteroid encounter
  - 2 comet encounters (in extended mission)
- Tight schedule
  - 34 months from project start to launch (30 planned)

DS1 Flight Software Development Conversion to Pathfinder Code

- Schedule problems forced major descope in flight software at launch minus 19 months
- Remote Agent AI software dropped from primary control of spacecraft
- 3-D Stack flight computer dropped
- Mars Pathfinder software adapted to DS1
  - Source code
  - Build environment, compilers, Makefiles, workstations
  - Associated tools
    - dwn_listen/pkt_show
    - Uplink product generation
  - Ground system

DS1 Flight Software Development Summary

- Just barely enough time to finish by launch
  - 4 month slip due to hardware & software not ready
  - Team worked extremely hard, efficiently
  - Deferred development required to meet launch date
  - Final software load built 1 month before launch
  - Final hardware modification 3 weeks prior to launch
- Plans were made to continue development during ops
  - M3 (Oct 1998) Launch load
  - M4 (Feb 1999) Bug fixes, some deferred development
  - M5 (Apr 1999) Remote Agent Experiment
  - M6 (Jun 1999) Deferred development for encounter
  - M7 (Apr 2000) Bug fixes, extended mission encounters

DS1 Flight Software Architecture

- C programming language
- vxWorks operating system
- Priority based context switching among tasks
- Message passing between tasks
  - FIFOs
  - Fixed queue depth
  - Extremely limited passing of data structures directly
- Health monitor
- Event Reporting (EVRs)
  - EVR_INFO, EVR_WARNING, EVR_FAULT, EVR_FATAL
  - Event time and location
  - Stack trace

Requirements Flow for DS1 Flight Software

- Informal, generated mostly from within FSW team
- Message parties
  - Entire team present
  - All messages reviewed for system interactions
- Problem statements
  - Summarizes what a software module is supposed to do
- Testing requirements document
  - Developed by FSW team based on problem statements, interviews, command dictionary
  - Includes test objectives

DS1 Flight Software Code Development

- Limited automated code generation used effectively
  - Fault protection (StateFlow, in-house perl scripts)
  - ACS (in-house perl scripts to generate headers, telemetry)
  - FSC (lisp code to generate code templates, message wrappers)
  - EVRs (yacc/lex tool to extract text strings, create ground code)
- Configuration management using CVS
  - Limited use of branching, concurrent work
  - Repository restricted to local homogeneous network
- Compilation, linking on AIX native code compiler
  - Only by integration engineer; developers had limited access
- Unit testing on single board computers (68040, PPC)
  - Green Hills, GNU compilers caught bugs AIX didn't

DS1 Flight Software Integration

- Module Acceptance Test (MAT)
  - Short set of requirements (a few pages)
  - Usually done by developers (except a few areas, spot checks)
- Software was integrated on Papabed prior to delivery
  - Flight-like configuration contains CPU, some EMs
  - Test coverage tracked by spreadsheet; PRs using GNATS
- Integration and delivery done by 2-man team
  - All deliveries traceable to a tagged version of source code. Build name built into code.
  - All deliveries built from clean checkout and build (limited exceptions)
  - Standard delivery location and naming convention
  - Delivery document emailed to test team

DS1 Flight Software Testing Approach

- Two testbeds (Testbed and Hotbench) use flight spares, EM's. Testbeds are located in cleanroom
- Spacecraft simulation runs in separate chasses
- Formal procedures to operate, log power cycles, mods
- Ground system can be used (default) or bypassed
- Every command sent to spacecraft is first tested on Testbed or Hotbench with limited exceptions
- Tracking of test results using spreadsheets
- Problems are logged using PR system derived from Pathfinder

In-Flight Experience with DS1 Flight Software

- In general, flight software has worked extremely well
- Four unintentional reboots have occurred to date
  - One due to operational error (reboot of Testbed went unnoticed)
  - One unknown cause (probably soft hardware problem in upl board)
  - Two reboots due to array bounds violations
- Several software incidents have required intervention
  - Twice overflowed queues have resulted in dropped messages, breaking handshakes (adherence to MAT requirements would have prevented these)
  - Race condition broke experiment software (inadequate testing did not adequately test context switching)
  - File system filled up (inadequate limit checking in code resulted in a huge file being generated)

DS1 Flight Software Performance

- Flight software provides on-board performance monitoring
  - 100 second history of task switches (must be turned on via command)
  - Idle time within EHA sampling period
  - Memory usage, stack high water marks
  - Open file descriptors
- Performance has been adequate
  - Approximately 50% processor margin during nominal operation
  - A number of tasks take all remaining CPU when they are active
    - Several operations require careful consideration of performance
    - Encounter sequence required considerable tweaking from performance point of view
  - State of telemetry buffer has biggest effect on nominal performance (size of free list)

Problems in the DS1 Software Development Process

- Software deliveries were usually later than initial estimates
  - Software development effort was underscoped
  - Personnel were burned out
  - Development team was too busy to document or help test
  - Integration and testing time got squeezed
  - Project was late to accept descope, contributing to team stress and further diluting their efforts
- Hardware was developed concurrently with software
  - A lot of time was wasted chasing problems that were caused by hardware instead of software
  - Hardware problems had to solved to continue development
- Hardware documentation was not always accurate
  - Finding hardware problems at integration time is very expensive

DS1 Flight Software Lessons Learned

- Inherit as much as possible , including personnel
- Live within your means
  - Minimize number of tools, platforms, methods, configurations, etc.
  - Be able to support what you use and develop
- Architect the software with multiple levels of self-checking and self-diagnosis and means for intervention
- Conduct independent code reviews
  - DS1 was too resource/schedule constrained
  - Reviews of troublesome code areas are currently underway and are finding many problems
- Use tools for static code analysis (e.g. array bounds)
- Operate the spacecraft conservatively; be paranoid

JPL's Next Generation Software Development Process

- Mission Data System (MDS) is JPL program to develop software and processes for future JPL missions
- Emphasis is on tools to assist development process
  - Source code instrumentation
  - Test management tools
  - Problem reporting and tracking tools
  - Test case generators
  - Source code analyzers
  - Test result analyzers
- MDS will deliver reusable code
  - C++
  - UML

MDS Source Code Instrumentation Strategy

- Event Logging Facility (ELF)
  - Similar to DS1 EVR's
  - Built-in mechanism for switching on, off, throttling ELF's
- Software Assertion Mechanism (SAM)
  - Inline boolean assertion
  - Data-driven boolean assertion
  - Timed temporal assertion
  - Output through ELF mechanism

MDS Test Management Systems

- Manage test cases created with heterogeneous set of test generators
- Provide configuration control and problem reporting system with open repository
- Provide scripting language for operating tests
- Provide for customization of tests with specific modules and versions
- Three tools are being evaluated
  - TestExpert (Silicon Valley Networks)
  - TestDirectory (Mercury Interactive)
  - CppUnit, Junit (Kent Beck, Erich Gamma, Martin Fowler)

MDS Problem Reporting and Tracking

- Track key info about reported issues
- Offer project-level summary reporting
- Support multiple types of users (manager, tester, end user)
- Coexist with MDS tools including configuration management, test management system, DocuShare, etc.
- Three tools are under consideration
  - ClearDDTS (Rational)
  - Team Track (Teamshare, being used by Interferometry Project)
  - AAMS (TMOD)

MDS Test Case Generators

- Produce parametric test vectors from given parameter-value tables
- Cover all pair-wise or n-wise (desire) parameter combinations
- Technical challenge to avoid combinational explosion in the number of test vectors generated
- Two tools are under consideration
  - AETG (Telcordia)
  - TCG (in-house, prototype exists)

MDS Source Code Analyzers

- Profilers (runtime performance)
  - Windview (NuMega)
  - DevPartner (NuMega)
- Syntax/Style checkers
  - QA C++ (Programming Research)
  - Code Wizard (Parasoft)
- Bounds checkers
  - Purify (Rational)
  - Insure (Parasoft)
  - CodeTest (WRS)
- Coverage Analysis
  - PureCoverage (Rational)

MDS Source Code Analyzers (continued)

- Fault index measurement (stability measures)
  - EMA, PCA/FI, EVOLV (John Munson/Quest)
- Test effectiveness measurement
  - Turnstile (John Munson/Quest)
- Reliability, fault content and location estimators
  - CASRE

MDS Test Result Analyzers

- ELF viewer
  - Display logged ELF events in time sequence
  - Under development; similar to tools used in Remote Agent experiment
- State variable/event viewer
  - Finer grained ELF viewer with additional display of state variables
- Custom viewers
  - Application specific

Summary and Conclusions

- Deep Space One is an existence proof for cheaper/faster projects at JPL
  - Flight software has worked very well
  - Test coverage was small, dictates very conservative mission operation
  - Success is due to hard work of team (unsustainable)
- JPL is evaluating tools for development/analysis/test of flight software for future missions
  - Some of these tools would help prevent some of the problems that DS1 has experienced, speed development
  - We don't yet have experience on how well many of these tools work and how useful they are
  - We need to pick the most effective subset of these tools.

LOCKHEED MARTIN

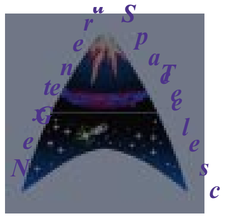
NGST Flight Software Testing

- Vision for the Future
  - Build Upon HST's Testing Success - "Specification" Based Testing Methodology
    - Not fully succeeded on HST
  - Test Development Concurrent with S/W Development (Specification/Requirements based testing)
  - Use CASE tools for Test Case Generators and Code analyzers (available now)
    - Test Cases build from Requirements During Code Design phase
    - Feed back Information from Code Analyzers (coverage, complexity) into test cases
  - Automated "Regression" Testing
  - Early H/W lab Development Essential

Improvements

- Need to Involve Test Personnel early in Software Life-Cycle
  - Have Them Write/Review Requirements
  - Make Better use of Requirements CASE Tools such as RTM
- Development of Test cases prior to code Development

LOCKHEED MARTIN



Top Problems/Issues

- Requirements Definition
  - Vague or Incomplete
  - Not Written for Testability
- Test Personnel Involved too Late in Life-Cycle
  - Usually included after design and into code development
- Tools are Inadequate

Flight Software Test Workshop

Eric Wentzel
Lockheed Martin Missiles and Space
Sunnyvale, Calif

eric.wentzel@lmco.com

Topics

LOCKHEED MARTIN
Missiles & Space

- External factors impacting FSW quality, cost, reliability
- Process
- Tools
- Architectures
- Test Environment

External Factors Impacting FSW Cost/Reliability

Lockheed Martin Missiles & Space

Lack of requirements stability

- Weak early system/mission engineering
- Slow evolution of GN&C solution (see above)
- Weak traceability, weak links to FSW and Test documentation
- Weak or simply wrong interface definitions
- Customer directed changes

Inadequate, declining budgets for test

- Cost growth early in a program squeezes time/money for adequate testing

Late hardware

- Compresses schedule available for FSW testing
- Test resource contention with other functional disciplines

Not enough time spent in test at either unit or integrated level

- GN&C solution late
- Requirements not understood or stable
- Shrinking budgets by time program gets to test phase

Process/Quality Issues

Lockheed Martin Missiles & Space

Standard processes that roughly don't change and are repeatable

- Repeatability often more important than new technology when it comes to quality
- How to balance new approaches/technologies with repeatability, confident solutions?

Strong attention early to quality of product (pay me a little now, a lot later...)

- Challenging entry and exit criteria for each phase of development and test
- Meaningful peer reviews with best domain experts (Make time!)
- Stronger attention to unit level testing

Object oriented designs pose benefits and challenges risk and defects

- OO designs challenging to develop and test
- Tools help (see next chart)

Tools need to follow process, not the other way around

Process/Quality Needs for Future

LOCKHEED MARTIN
Missiles & Space

Follow internal processes even when cost/schedule cause pressure to cut corners

- **Quality needs to be a focus at every step in the software development life cycle**
- **For Unit Test, Integrated Testing:**
  - Standards for test coverage
  - Tough peer reviews with correct domain experts
  - Enforce entry/exit criteria
  - Robust COTS Tools

OO Designs need to validated in Phase B

- **OO products reduce cost, risk and cycle time, increase reliability**
- **Mature product base is key to success with OO designs**

Tool Issues

LOCKHEED MARTIN Missiles & Space

New tools with lots of promise vs older tools with lots of users/reliability

- **Major risks associated with flashy, new tools that have a low installed base**
- **Managing risk means managing your COTS environment**
- **Backwards compatibility often all talk, no action**
- **Latest comprehensive Software Development Environments a major step forward to automated analysis and test**
  - Buyer beware - these are new and challenging to learn and absorb
  - Processor selection drives tools choices, content, maturity

C&DH hardware selections drive ultimate tool integration

- **The more commercial a hardware solution, the better the tool environment**
- **Software now more important, riskier, more costly than hardware - and should drive hardware selections, not the other way around**

Tool integration a major challenge due to

- **Complexity**
- **Pace of change in tools by vendors (CM, backward compatibility...)**
- **Immaturity of most new tools**
- **Data Base tools and CM still a major shortfall in recent system implementations**

Next Generation Processes/Tools: Needs for Future

Close coupling of tools and process, demonstrated early in a program's life cycle

Increased automation to achieve improvements

- Test coverage
- Reliability
- Cycle time reductions
- Cost savings

In-house simulation, real time and test command and control and data base solutions need to be robust, integrated and mature

- Early unit testing improvements needed - improved tools can help
- Need much stronger, early, high fidelity integrated testing in both non real time and real time environments
- Contractor proposed solutions must take out risk via IRAD, the programs need zero NRE test environments

Tools Needs for Future

Specific tools needs in future - linked to extent practical

- **Integrated or complementary tools**
- **Coverage tool**
- **Structure Analysis**
  - Automated peer reviews
  - What needs to be tested
- **Run time errors - memory and dynamic processes**
- **Metrics tools - conforming to stds**
- **CASE Tool OOA/OOD - generate code, interfaces**
- **CPU simulator**
- **Test management tools to link requirements to test scripts**
- **Automated reqmts traceability linked to all other FSW documentation**

Process needs

- **Strongly documented internal and external interfaces**
- **Con ops early**
- **Units and subsystems thoroughly tested before integration**
- **Simulations for NRT and RT testing need to be modular**

Next Generation Processes/Tools: Needs for Future

Select Flight Processor based on availability of FSW tools, in addition to suitability for the mission

- **Commercial processors have richest selection of tools**
  - More types of tools
  - More mature tools due to broader installed base

Prototype tool implementions early in program

- **Test drive process/tools mix while in competition**
- **Focus on quality when assessing selected process/tool mix**
  - Requirements coverage
  - Reliability
  - Repeatability
  - Strong analysis capabilities in selected products
- **Smooth flow from non-real time testing to real time testing**

Architecture Issues

Mature FSW OO architecture and reuse paradigm to:

- **shorten cycle time**
- **reduce cost/risk**
- **Improve quality**

Application engineering

- **Key to reuse success is strong application engineering process and tools**
  - High degree of automation
  - Tight coupling with requirements data base
  - Demonstrated success

Data system architecture very strong contributor to risk, cost, reliability

- **Known commercial architectures best for reliable solutions**

Architecture Needs for Future

Demonstrated OO architecture together with application engineering process and tools

- **Prototype (demo) application engineering part of Phase B**
- **Robust architecture evaluated, fine tuned (need to climb maturity curve)**

Data system architecture design needs to be straight forward, testable, Reliable

- **Strong reliance on commercial approaches**
- **Eliminate unwanted variabilities**

Tools need to be considered part of the architecture equation

- **Need tools architecture**
- **Need prototyping very early to confirm approach, interfaces, benefits**

Test Environment Issues

Test facilities for space systems very expensive, difficult to use

- Test facility resource contention always an issue
- Late maturation of facility often a problem
- Integration & Test Lab highly complex, penalizing inadequate unit or subsystem testing with long/expensive debug at system level

Low budget, or single shot programs tend to reduce the fidelity of the test environment to reduce cost, either before or after the program starts

Lack of automation in coupling test requirements to test preparation to test execution drives cost and reliability

Test Environment Needs for Future

Object orientation in C&DH and Test Environment can produce desired results

- **Software abstracted from hardware**
- **Innovative C&DH architectures**
  - Test cards individually with commercial, low cost hardware
  - Test cards individually in high fidelity sim environment with low cost commercial (not flight) processors

Develop like you test like you fly

- **GN&C analysis/sim environment same as FSW development/test environment same as operator training environment**
- **Data base and command/telemetry solution same from FSW development, FSW Test, Vehicle Test and Vehicle Ops**
  - Variabilities in Commercial, NASA, Military and Classified Command and Control solutions complicates prime contractor's job
- **High fidelity FSW Test Environment with GN&C components in the loop pays dividends during system test**
  - Sim needs to be capable of very rapid reconfiguration for sim object or hardware in the loop

Conclusions

Processes need to be strengthened, "test driven" and followed

- Reusable products change the equation
- Test approach needs to change accordingly
- Integrated tool will environment help

Tools need to be integrated with process, matured and repeatable results Demonstrated

- Major area for meeting customer desire for improvements

Innovative C&DH and test product architectures can reduce test cycle time, risk and cost

- But more development needed

FSW Test Workshop

Martha I. Chu

Johns Hopkins University
Applied Physics Laboratory
Laurel, Maryland 20732

Tel: (240) 228-7961
Martha.chu@jhuapl.edu

JHU/APL FSW Current Practices

- FSW developers realize that testing is important and necessary
- Common Test Environment used for development, validation, spacecraft I&T, and mission simulation.
- Two test efforts, development and I&T, are not complementary and require duplicate infrastructure; improved on current missions
- Dedicated fully implemented targets in the form of wire-wrap or engineering models used for development and testing
- Comprehensive emulation of target environments through detailed hardware-in-the-loop simulation.
- Analyst simulation and test without dependence on flight hardware

JHU/APL FSW Test Lessons Learned

- Project level definition of FSW testing requirements
  - Should be understood at the start of a project
  - Risks have to be addressed
- Well written requirements and carefully thought out design are a necessity
  - Need to be Clear, Concise, Complete and ***Testable***
  - Need to be Carefully considered
    - Unnecessary Complexity in Requirements and Design means unnecessary Complexity in Testing.
- Need test philosophy and approach defined
  - test process throughout the life cycle
  - multiple, incremental releases
  - strict configuration control

JHU/APL FSW Test Lessons Learned

- Test Equipment
  - Need dedicated access
  - Needs to be planned at project startup
- Common Test Environment
  - Improves Subsystem, validation, spacecraft I&T, and mission testing*
  - GSE needs to be planned from the start and mature before testing starts
- Analyst simulation and test approach* should extend to other FSW development efforts

* has been accomplished

FSW Test Workshop

JHU/APL vision for the next generation processes/tools for flight software testing, which can be achieved within a 2-3 year timeframe.

- Testing as an integral part of the project planning and design process
  - A philosophy, commitment, and process
  - Defined and cost at proposal/budget submission
- Lead Test Engineer responsible to Mission Software Engineer for all FSW testing
- A clearly defined and flexible process that includes:
  - A “model based” test concept
  - An “Outside-in” hierarchy of tests
  - IV&V requirements testing
  - Full traceability
  - Automation
  - Tools
- Project will commit to have early, testable and tightly controlled requirements

FSW Test Workshop

JHU/APL vision for the next generation processes/tools for flight software testing, which can be achieved within a 2-3 year timeframe.

- At the System Level: (C&DH, G&C, Instruments, etc.)
  - Integrate subsystems early and often
  - Testing covered by the Lead Test Engineer
- At the Subsystem Level:
  - carried out by test engineers under authority of Lead Test Engineer
  - simple, comprehensive, repeatable test procedures
  - early and multiple releases

FSW Test Workshop

JHU/APL vision for the next generation processes/tools for flight software testing, which can be achieved within a 2-3 year timeframe.

- At the Component Level:
  - carried out by developers within their own jurisdiction of responsibility*
  - Results logged and archived for traceability
  - Wide use of pre-tested components
  - Built-in test capability
- IV & V Testing
  - Independent*
  - Dedicated resources**
  - Requirements testing; Currently scenario testing
  - Documented, traceable, and repeatable testing**

*has been accomplished

** somewhat accomplished

FSW Test Workshop

JHU/APL vision for the next generation processes/tools for flight software testing, which can be achieved within a 2-3 year timeframe.

- Test Equipment and Environments
  - A common subsystem, IV&V, and spacecraft test environment*
  - Dedicated access to high fidelity test equipment and targets for subsystem, IV&V, and spacecraft teams**
  - Enhanced by COTS tools and simulators to off-load common development environment
  - Reusable from mission to mission
  - GSE needs to be validated prior to use
- Regression Testing
  - Part of the process
  - Planned
  - Leverage other testing efforts at all levels

*has been accomplished

** somewhat accomplished

FSW Test Workshop

What you see as necessary to accelerate the testing process compared with your current methods/systems. Please address all aspects of test phases.

- Organizational shift to recognize the importance of software testing
- Define test process that captures roles and responsibilities
- Modeling tools to allow early testing of architectures and high level designs - model based (design) testing
- Reusable test methods and tools for bottom-up testing, starting with software unit testing
- Automation of the test process using tools
- Education, all FSW developers and test engineers need training in the testing process, methods, approaches, etc
- Writing requirements for testability

FSW Test Workshop

What you see as necessary to accelerate the testing process compared with your current methods/systems. Please address all aspects of test phases.

- Active involvement in processor selection
  - use primitive processors drives barbaric test environment
- Commitment to repeatable testing process
- Common test environment*
- Dedicated test environments**
- Traceability of Unit, Subsystem, I&T, and IV&V Testing
- Test Readiness Reviews as part of the managed process

*has been accomplished

** somewhat accomplished

FSW Test Workshop

Highlight the top 2-3 problems or issues, which dramatically affect cost and schedule.

- Testing is an afterthought
  - Lack of top down test tools delays detection of high level architecture/design problems, requiring major rework at lower levels
  - Lack of test-oriented development process promotes increasing cost and schedule
  - Lack of top down test-oriented planning and development makes it difficult to coordinate development schedules for different components
- Little FSW reuse has been accomplished
- Hardware and software delays result in testing being de-scoped

FSW Test Workshop

Highlight the top 2-3 problems or issues, which dramatically affect cost and schedule.

- Reliance on hardware availability at all levels of testing
- Minimum usage of available tools
- Lack of a common, mature GSE from mission to mission